UCRL-TM-229498

RadSrc/Monte Carlo Code Interface Manual

Larry Hiller, Tom Gosnell, Jeff Gronberg, Doug Wright[*]
Lawrence Livermore National Laboratory

February 16, 2007

Abstract

RadSrc ("Rad-Source") is a library for calculating gamma ray distributions. An initial material specification is aged and the daughter isotopes calculated to create the complete spectrum. RadSrc can be linked into, initialized, and called from other programs. This document specifies how to do this in GEANT4, COG and MCNP(X). The software library and interface examples can be downloaded from `http://nuclear.llnl.gov/simulation`.

1 Configuring GEANT4 to use RadSrc

The user will need to have access to a GEANT4 installation. It is assumed that the user can already run a GEANT4 job. Download the RadSrc library to your computer. In the src directory type gmake in order to create the libradsrc.a file. *It is highly recommended that you determine what compiler the GEANT4 installation is using and modify the RadSrc Makefile to use the same compiler.*

The `geant` directory in the RadSrc release contains a sample GEANT4 job which accesses the radsrc library and runs a 10kg uranium ball problem. Type `source setup` to create the RadSrc environment variables needed by the Makefile and the code. If you have GEANT4 installed on your system and the GEANT4 environment variables are set then you should be able to type `gmake` to create an executable. An executable file called `exampleN01` should be located in the `bin` directory.

Type `bin/`*[system type]*`/exampleN01 <example.in` to run the program.

The GEANT4 GNU makefile has been modified to link in the RadSrc libraries through the addition of two line:

```
EXTRALIBS += -L$(RADSRC_HOME)/lib/ -lradsrc
CPPFLAGS += -I$(RADSRC_HOME)/src/libradsrc/
```

The environment variable `RADSRC_HOME` is defined in the setup routine along with the variable `RADSRC_LEGACYDATA` which points to the RadSrc data files.

[*]Contact info: wright20@llnl.gov, 925-423-2347

2 Calling RadSrc from within GEANT4

The RadSrc routines are called from within the `ExN01PrimaryGeneratorAction` class which handles event generation for the problem. In the header files we include the RadSrc header files:

```
#include "radsource.h"
#include "cpp_api.h"
```

and create some pointers variables that will point to instances of the RadSrc class,

```
    radsrc::CRadSource* pRadSource;
    radsrc::CRadSource* t1RadSource;
    radsrc::CRadSource* t2RadSource;
```

the RadSrc routines live in the namespace `radsrc::` in order to prevent conflicts with geant4 classes.

To create a new instance of RadSrc one calls,

```
  t1RadSource = radsrc::CApi::newSource();
```

The instance of RadSrc must then be initialized with a problem definition. A number of ways to do that are provided. One way is to create a text file with the problem definition and pass the location of that file to the `LoadConfig` member function.

```
  t1Good = radsrc::CApi::loadConfig(t1RadSource, (const std::string) FileName);
```

If a NULL string is passed the program will look for the environment variable `RADSRC_CONFIG` to find the input text file.

The text file must be in the form of,

```
U235     90.0
U238     10.0
AGE      20.0
```

where the fraction of each isotope is specified and should add up to 100%. To allow for problems with contamination the fractional sum can be slightly greater than 100%. The last line in the file should be an AGE card with the age given in years.

Additionally, the problem specification can be passed as input lines to the code,

```
  radsrc::CApi::addConfig(t2RadSource, ‘‘U235 90.0’’);
  radsrc::CApi::addConfig(t2RadSource, ‘‘U238 10.0’’);
  radsrc::CApi::addConfig(t2RadSource, ‘‘AGE 20.0’’);
  t2Good = radsrc::CApi::sourceConfig(t2RadSource);
```

where the `addConfig` member function accepts text lines of input and the `sourceConfig` processes the input and performs the RadSrc calculations and setup.

Once the problem is specified the gamma-ray distributions can be sampled with a call to:

```
  G4double energy = radsrc::CApi::getPhoton(pRadSource, localran ) * keV;
```

The function returns an energy in keV. The function `localran` is a wrapper for the standard GEANT4 random number generator `G4UniformRandom`.

The RadSrc problem can also be specified from the GEANT4 command line through the commands defined in the `ExN01PrimaryGeneratorMessenger` class. They duplicate from the command line what is available in the code.

```
/radsrc/file './problem.in'
```

reads the problem definition from the specified file

```
/radsrc/file ''
```

will look for the environment variable `RADSRC_CONFIG` to find the input text file. The complete problem specification can also be passed on the command line by using

```
/radsrc/input U235 90.0
/radsrc/input U238 10.0
/radsrc/input AGE 20.0
/radsrc/update
```

The GEANT4 code is set up so that the `ExN01PrimaryGeneratorAction` class will try to find a RadSrc input text file at instanciation. A constructor function is provided which passes a string which specifies the file location. See `exampleN01.cc` for an example. That definition can be overridden from the command line. If no problem definition is specified the program will terminate at the first event.

3 Calling RadSrc from within FORTRAN

COG and MCNP(X) provide dummy source subroutines called IsoP.F and source.F90, respectively, which can be used to call the RadSrc routines. While RadSrc is native C++, a number of Fortran callable subroutines are provided to allow Fortran code to use the package. All routines names begin with `RS` in order to prevent accidental conflicts with the Monte Carlo code. Multiple instances of the RadSrc library can be created to model multiple sources. Simply provide a different handle for each case. Inside the subroutine define the variables and functions that the library will use.

```
  INTEGER*8 HANDLE
  LOGICAL SUCCESS, RSLOADCONFIG, FIRST, RSSOURCECONFIG
  REAL*8 RSGTRPHOTON

  EXTERNAL RSLOADCONFIG, RSGTRPHOTON, RSADDCONFIG, RSSOURCECONFIG

  COMMON /rscommon/ HANDLE, FIRST
  DATA first /.true./
```

During the first pass through the subroutine an instance of the RadSrc library is created and the problem specified.

```
  IF(FIRST)THEN
    FIRST=.FALSE.
    CALL RSNEWSOURCE(HANDLE)
    SUCCESS = RSLOADCONFIG(HANDLE,'./problem.in')
    IF (SUCCESS .NEQV. .TRUE.) THEN
        PRINT *, 'Error setting up problem'
        STOP
    ENDIF
  endif
```

The subroutine `RSNEWSOURCE(HANDLE)` creates an instance of the RadSrc library and returns a pointer to it. The specification of the problem is held in a text file of the form:

```
U2325    90.0
U238     10.0
AGE      20.0
```

where the fraction of each isotope specified and should add up to 100%. To allow for problems with contamination the fractional sum can be slightly greater than 100%. The last line in the file should be an AGE card with the age given in years. The subroutine `RSLOADCONFIG(HANDLE,'./problem.in')` reads in the specified problem definition file and performs the problem calculations and setup. It returns `.FALSE.` if there is a problem. If `RSLOADCONFIG(HANDLE,'')` is specified with a null string then the code will check the environment variable

```
setenv RADSRC_CONFIG [path to text file with problem specification]
```

for the location of the input file.

Alternatively, the problem specification can be passed as strings using the `RSADDCONFIG` and `RSSOURCECONFIG` subroutines as shown:

```
     CALL RSADDCONFIG(HANDLE,'U234 0.00071')
     CALL RSADDCONFIG(HANDLE,'U235 0.182')
     CALL RSADDCONFIG(HANDLE,'U236 0.00284')
     CALL RSADDCONFIG(HANDLE,'U238 99.814')
     CALL RSADDCONFIG(HANDLE,'AGE 15')
     SUCCESS = RSSOURCECONFIG(HANDLE)
```

Finally, after the problem has been set up, the user can call the subroutine `RSGTRPHOTON(HANDLE,RNG)` to sample the photon distribution and return an energy value. The native random number generator is passed to the subroutine so that the Monte Carlo code can maintain control over the random number sequence. The subroutine returns an energy value in keV.

4 Configuring COG to use RadSrc

The user will need to have access a COG installation. Download the RadSrc library to your computer. In the src directory type gmake in order to create the libradsrc.a file. *It is highly recommended that you determine what compiler the COG installation is using and modify the RadSrc Makefile to use the same compiler.*

COG provides the capability to compile a user source routine and dynamically link it into the COG executable. Make a directory containing the makefile `COGUserlib.make` and the user source subroutine `IsoP.F` which can be found in the `usrdet` directory of your COG release. The makefile must be modified to link in the RadSrc library. Modify the `LDOPTS` variable to add:

```
LDOPTS = ... -L$(RADSRC_HOME)/lib/ ... -lradsrc -lstdc++
```

This is correct for the intel compiler. Other compilers may require different libraries. The environment variable `RADSRC_HOME` should be set to point to your installation of RadSrc. As the RadSrc library is written in C++ one must also link in the `stdc++` library.

At this point you can try to link the COG user library to ensure that the libraries are being properly linked. In order for the RadSrc library to be able to find its data files an environment variable must be set:

```
setenv RADSRC_LEGACYDATA $(RADSRC_HOME)/data/
```

5 Options Unique to COG

The IsoP.F subroutine can also be configured to accept input from the COG input file. Using the code fragment:

```
  IF(FIRST)THEN
    FIRST=.FALSE.
    CALL RSNEWSOURCE(HANDLE)
    DO 99 I=1,NARGS
      CALL RSADDCONFIG(HANDLE,ARGA(I))
 99 ENDDO
    SUCCESS = RSSOURCECONFIG(HANDLE)
    IF (SUCCESS .NEQV. .TRUE.) THEN
        PRINT *, 'Error setting up problem'
        STOP
    ENDIF
  endif
```

the COG user source input will be passed to the RadSrc library. Setup up the source specification in the COG input file with the following format:

```
source
  usrsor IsoP
    U235 90.0
    U238 10.0
    AGE  15.0
```

Due to the limitations of the COG input parser none of the individual words in the input can be longer than 8 characters.

6 Configuring MCNP to use RadSrc

The user will need to have access to and be able to recompile the MCNP source code. Download the RadSrc library to your computer. In the src directory type gmake in order to create the libradsrc.a file. *It is highly recommended that you determine what compiler the MCNP installation is using and modify the RadSrc Makefile to use the same compiler.*

The MCNP Makefiles must be modified to link in the RadSrc libraries. In the Source directory there is a master makefile and in the config directory there are platform specific makefiles. In the appropriate makefile for your installation add a line:

```
EXTRALIBS = -lstdc++ -L$(RADSRC_HOME)/lib/ -lradsrc
```

As the RadSrc library is written in C++ one must also link in the stdc++ library. To allow the program to see the class definitions add a path to the C++ headers by modifying

```
INCLUDE_DIRS = -I$(RADSRC_HOME)/src/libradsrc/
```

Ensure that the environment variable `RADSRC_HOME` points to your copy of the RadSrc installation. At this point you can try to relink the MCNP executable to ensure that the libraries are being properly linked. In order for the RadSrc library to be able to find its data files an environment variable must be set:

```
setenv RADSRC_LEGACYDATA $(RADSRC_HOME)/data/
```